实验九　VGA 接口控制器实现

2020 年秋季学期

杨飞洋 191220138

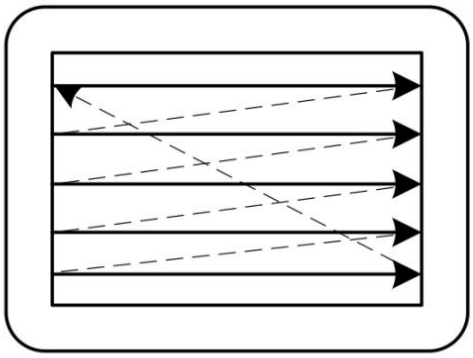
目录

1. 实验目的：

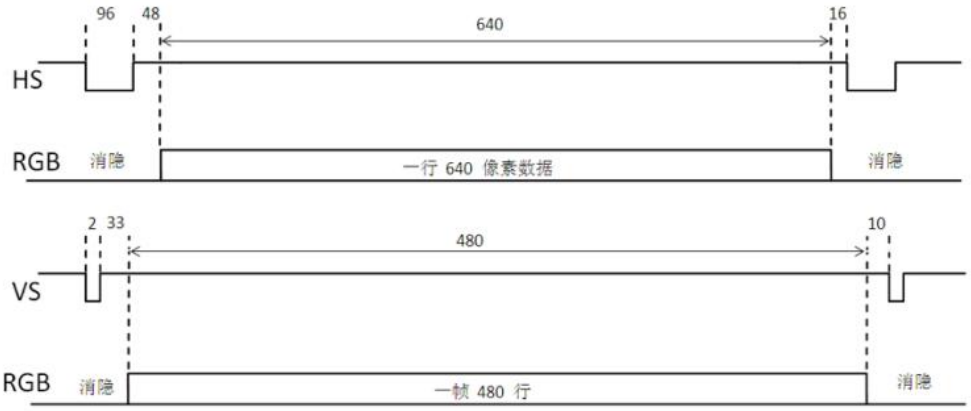
学习 VGA 接口原理，学习 VGA 接口控制器的设计方法。
设计工程控制显示器上的显示。

2. 实验原理（知识背景）

图像的显示是以像素（点）为单位，显示器的分辨率是指屏幕每行有多少个像素及每帧有多少行，标准的 VGA 分辨率是 640×480，也有更高的分辨率， 如 1024×768、1280×1024、1920×1200 等。从人眼的视觉效果考虑，屏幕刷新的频率（每秒钟显示的帧数）应该大于 24，这样屏幕看起来才不会闪烁，VGA 显示器一般的刷新频率是 60HZ。
每一帧图像的显示都是从屏幕的左上角开始一行一行进行的，行同步信号是一个负脉冲，行同步信号有效后，由 RGB 端送出当前行显示的各像素点的 RGB 电压值，当一帧显示结束后，由帧同步信号送出一个负脉冲，重新开始从屏幕的左上端开始显示下一帧图像，如下图所示。



在标准的 640 × 480 的 VGA 上有效地显示一行信号需要 96+48+640+16=800 个像素点的时间，其中行同步负脉冲宽度为 96 个像素点时 间，行消隐后沿需要 48 个像素点时间，然后每行显示 640 个像素点，最后行消隐前沿需要 16 个像素点的时间。所以一行中显示像素的时间为 640 个像素点时间，一行消隐时间为 160 个像素点时间。
在标准的 640 × 480 的 VGA 上有效显示一帧图像需要 2+33+480+10=525 行时间，其中场同步负脉冲宽度为 2 个行显示时间，场消隐后沿需要 33 个行显示时间，然后每场显示 480 行，场消隐前沿需要 10 个行显示时间，一帧显示时间 为 525 行显示时间，一帧消隐时间为 45 行显示时间。
因此，在 640 × 480 的 VGA 上的一幅图像需要 525 × 800 =420k 个像素点的时间。而每秒扫描 60 帧共需要约 25M 个像素点的时间。



开发板和 ADV7123 芯片之间的接口引脚包括 3 组 8bit 的颜色信号 VGA_R[7:0], VGA_G[7:0], VGA_B[7:0]，行同步信号 VGA_HS，帧同步信号 VGA_VS，VGA 时钟信号 VGA_CLK，VGA 同步（低有效）VGA_SYNC_N，和 VGA 消隐信号（低有效）VGA_BLANK_N。

3. 实验环境/器材

系统：windows10
开发软件：Quartus 17.1 Lite
开发板：DE10 Standard
仿真环境：ModelSim
芯片：Cyclone V ， 5CSXFC6D6
显示器：标准 VGA 显示器（640*480）

4. 设计思路

基础模块：
通用时钟生成模块：

```
module clkgen(
   input clkin,
   input rst,
   input clken,
   output reg clkout
);
parameter clk_freq=1000;
parameter countlimit=50000000/2/clk_freq; // 自动计算计数次数
reg[31:0] clkcount;
always @ (posedge clkin)
   if(rst)begin
      clkcount=0;
      clkout=1'b0;
   end
   else begin
      if(clken)begin
         clkcount=clkcount+1;
         if(clkcount>=countlimit)begin
            clkcount=32'd0;
            clkout=~clkout;
         end
         else
            clkout=clkout;
      end
      else begin
         clkcount=clkcount;
         clkout=clkout;
      end
   end
endmodule
```

加一行如下的代码就可以生成 VGA_CLK

```
clkgen #(25000000) my_vgaclk(clk,1'b0,1'b1,signal);
```

时钟驱动生成信号模块：

```
module vga_ctrl(
    input pclk, //25MHz 时钟
    input reset, //置位
    input [23:0] vga_data, //上层模块提供的 VGA 颜色数据
    output [9:0] h_addr, //提供给上层模块的当前扫描像素点坐标
    output [9:0] v_addr,
    output hsync, //行同步和列同步信号
    output vsync,
    output valid, //消隐信号
    output [7:0] vga_r, //红绿蓝颜色信号
    output [7:0] vga_g,
    output [7:0] vga_b
);

//640x480 分辨率下的 VGA 参数设置
parameter h_frontporch = 96;
parameter h_active = 144;
parameter h_backporch = 784;
parameter h_total = 800;

parameter v_frontporch = 2;
parameter v_active = 35;
parameter v_backporch = 515;
parameter v_total = 525;

//像素计数值
reg [9:0] x_cnt;
reg [9:0] y_cnt;
wire h_valid;
wire v_valid;

always @(posedge reset or posedge pclk) //行像素计数
    if (reset == 1'b1)
        x_cnt <= 1;
    else begin
        if (x_cnt == h_total)
            x_cnt <= 1;
        else
            x_cnt <= x_cnt + 10'd1;
    end
```

```
always @(posedge pclk) //列像素计数
    if (reset == 1'b1)
        y_cnt <= 1;
    else begin
        if (y_cnt == v_total & x_cnt == h_total)
            y_cnt <= 1;
        else if (x_cnt == h_total)
            y_cnt <= y_cnt + 10'd1;
    end

//生成同步信号
assign hsync = (x_cnt > h_frontporch);
assign vsync = (y_cnt > v_frontporch);

//生成消隐信号
assign h_valid = (x_cnt > h_active) & (x_cnt <= h_backporch);
assign v_valid = (y_cnt > v_active) & (y_cnt <= v_backporch);
assign valid = h_valid & v_valid;

//计算当前有效像素坐标
assign h_addr = h_valid ? (x_cnt - 10'd145) : {10{1'b0}};
assign v_addr = v_valid ? (y_cnt - 10'd36) : {10{1'b0}};

assign vga_r = vga_data[23:16];
assign vga_g = vga_data[15:8];
assign vga_b = vga_data[7:0];

endmodule
```

第一个任务：显示不同颜色的条纹，主要思路是根据 h_addr 的大小来控制输出的 data

模块的输入：

```
module stripe(
    input clk,
    input reset,
    input clken,
    output HS,
    output VS,
    output blank,
    output vga_clk,
    output [7:0]red,
    output [7:0]green,
    output [7:0]blue
);
```

clk 为 50MHz 的时钟信号，reset 置位键，直接赋 0，clken 使能赋 1，HS、VS 输出行同步、列同步信号，blank 输出消隐信号，vga_clk 输出 VGA 时钟，red、green、blue 为颜色信息。

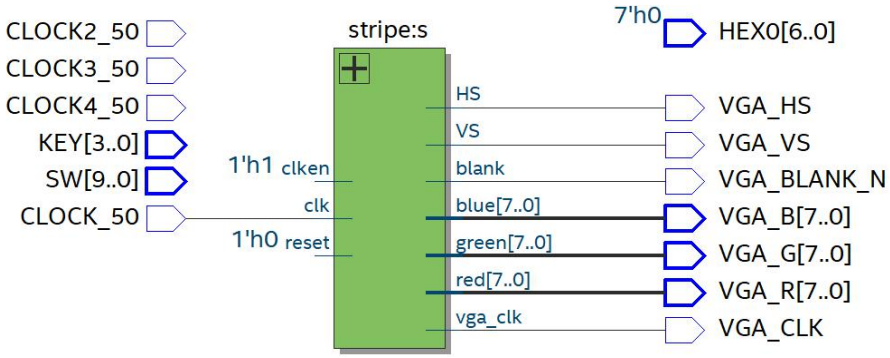
首先利用 clkgen 模块生成所需的 VGA_CLK，根据 haddr 的值来决定 data 的值，以下为模块代码:

```
reg [23:0]data = 24'b0;
wire [9:0]h_addr;
wire [9:0]v_addr;
wire signal;

assign vga_clk = signal;

clkgen #(25000000) my_vgaclk(clk,1'b0,1'b1,signal);
vga_ctrl v(signal,1'b0,data,h_addr,v_addr,HS,VS,blank,red[7:0],green[7:0],blue[7:0]);

always @(posedge vga_clk) begin
   if(h_addr <= 200)
      data = 24'hFF0000;
   else if(h_addr <= 400)
      data = 24'h00FF00;
   else
      data = 24'h0000FF;
end
```

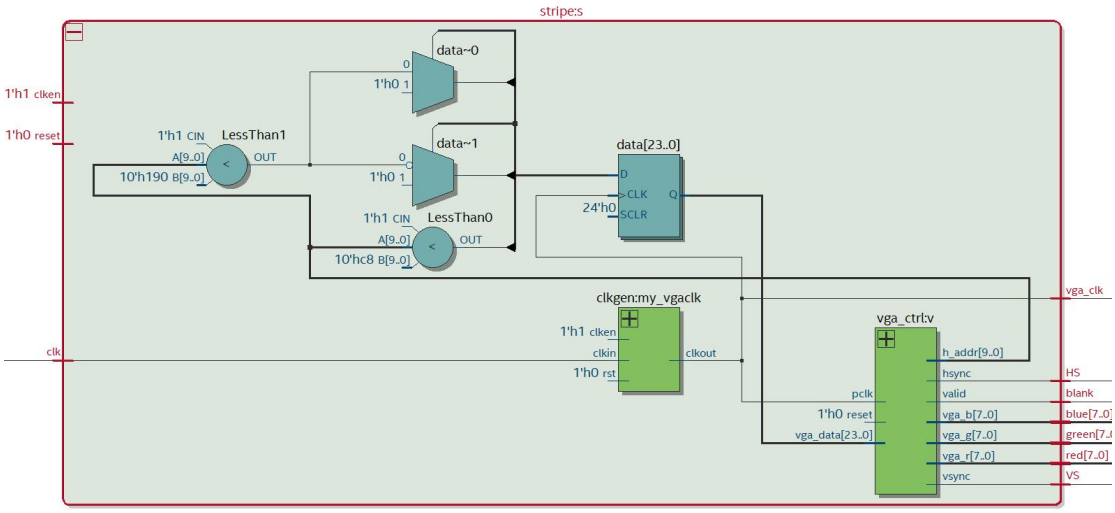
在总文件中的调用:

```
stripe s(CLOCK_50,1'b0,1'b1,VGA_HS,VGA_VS,VGA_BLANK_N,VGA_CLK,VGA_R,VGA_G,VGA_B);
```

总的 RTL viewer:



stripe 模块的 RTL viewer:



第二个任务：显示一张图片（已有 mif 文件）
主要思路：用 rom 存储 mif 文件内容，将 vga_ctrl 返回的 h_addr 和 v_addr 拼接成地址返回 ram 读取数据

模块输入和之前相同

```
module picture(
    input clk,
    input reset,
    input clken,
    output HS,
    output VS,
    output blank,
    output vga_clk,
    output [7:0]red,
    output [7:0]green,
    output [7:0]blue
);
```

模块主要的逻辑代码如下图：
data 是要输入 vga_ctrl 模块的数据，它由 h_addr 和 v_addr 控制，且每个颜色只取高四位。
temp_color 存放从 rom 中读取的 12bit 数据
signal 还是输出的 vga 时钟信号
addr 就是输入进 rom 的地址，用于读取数据
h_addr 和 v_addr 就不用多说了。

调用 clkgen 模块生成所需的 VGA 时钟信号，调用 vga_ctrl 模块根据 data 读取数据输出红蓝绿三个颜色信息，再输出 h_addr 和 v_addr，在 always 语句中将 h_addr 和 v_addr 拼接成 addr 输入进 rom 模块，rom 模块输出 temp_color，temp_color 再给到 data，这样循环往复，就能持续生成这张图片。

```
reg [23:0]data;
wire [9:0]h_addr;
wire [9:0]v_addr;
wire signal;
wire [11:0]temp_color;
reg [18:0] addr=19'b0;

assign vga_clk = signal;

clkgen #(25000000) my_vgaclk(clk,1'b0,1'b1,signal);
vga_ctrl v(signal,1'b0,data,h_addr,v_addr,HS,VS,blank,red[7:0],green[7:0],blue[7:0]
rom r(addr,signal,temp_color);
//my_rom cat(addr,signal,temp_color);

always @(posedge vga_clk) begin
    addr = {h_addr,v_addr[8:0]};
    data[23:20] = temp_color[11:8];
    data[19:19] = 4'b0;
    data[15:12] = temp_color[7:4];
    data[11:8] = 4'b0;
    data[7:4] = temp_color[3:0];
    data[3:0] = 4'b0;
end
```

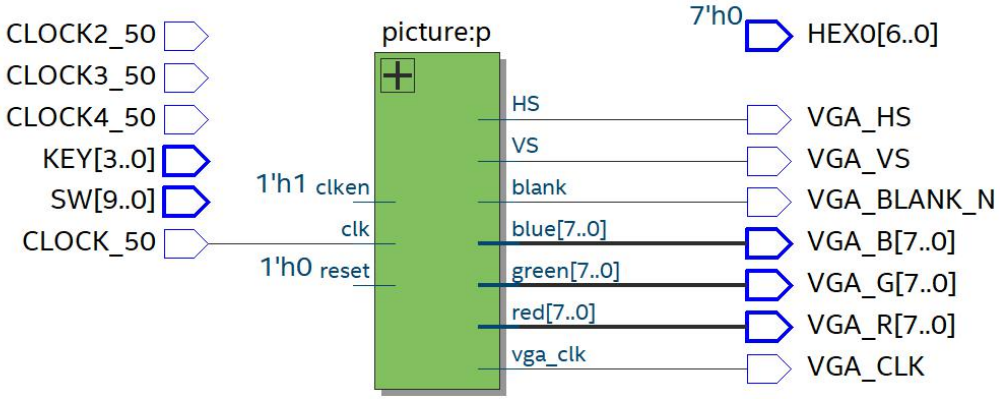
rom.v 文件信息：

```
altsyncram_component.address_aclr_a = "NONE",
altsyncram_component.clock_enable_input_a = "BYPASS",
altsyncram_component.clock_enable_output_a = "BYPASS",
altsyncram_component.init_file = "picture.mif",
altsyncram_component.intended_device_family = "Cyclone V",
altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=NO",
altsyncram_component.lpm_type = "altsyncram",
altsyncram_component.numwords_a = 327679,
altsyncram_component.operation_mode = "ROM",
altsyncram_component.outdata_aclr_a = "NONE",
altsyncram_component.outdata_reg_a = "CLOCK0",
altsyncram_component.widthad_a = 19,
altsyncram_component.width_a = 12,
altsyncram_component.width_byteena_a = 1;
```

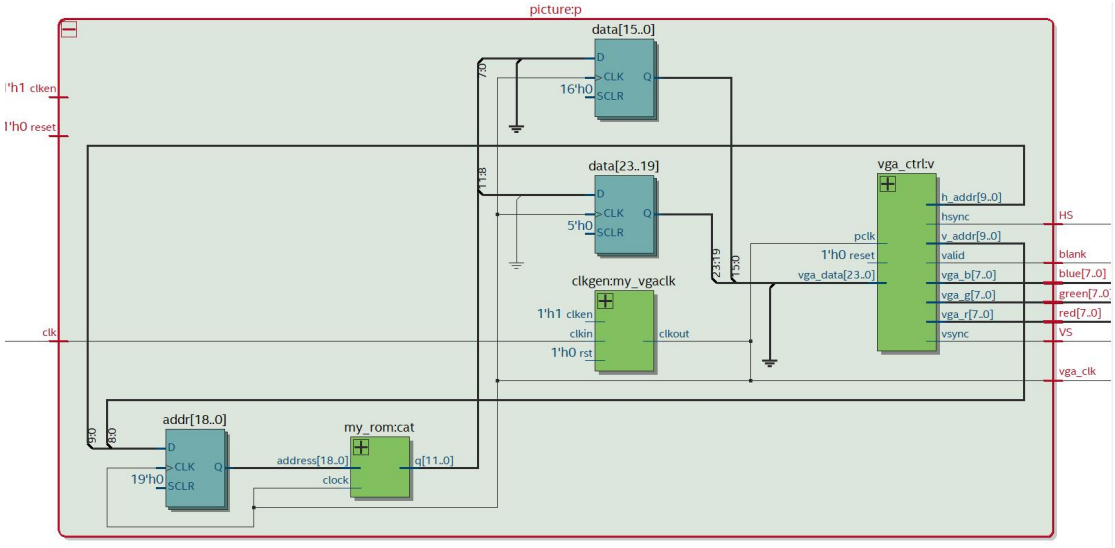
在总模块中的调用：

```
assign VGA_SYNC_N = 0;
picture p(CLOCK_50,1'b0,1'b1,VGA_HS,VGA_VS,VGA_BLANK_N,VGA_CLK,VGA_R,VGA_G,VGA_B);
```

总的 RTL viewer：



picture 模块的 viewer：

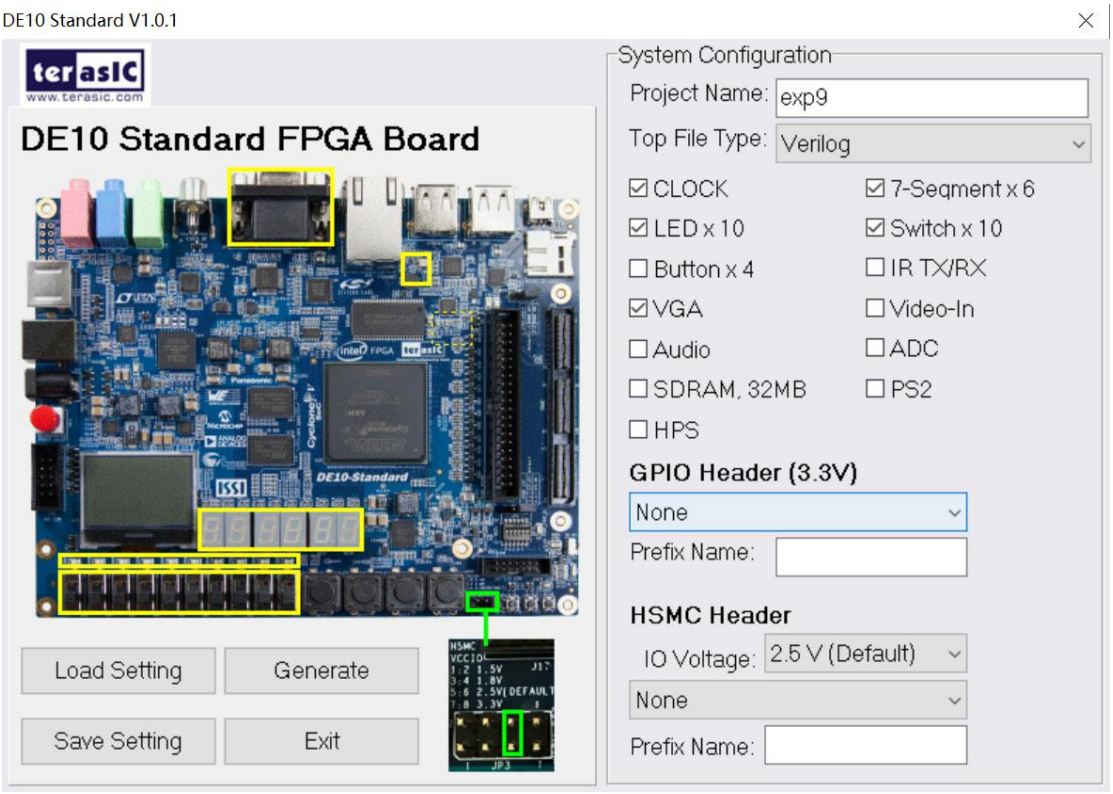


第三个任务：自己生成一张图片的 mif 文件并输出
思路：和第二个任务唯一不同的就是要生成这张图片的 mif 文件，之后所有的操作都是一样的。考虑用 matlab 工程生成 mif 文件。

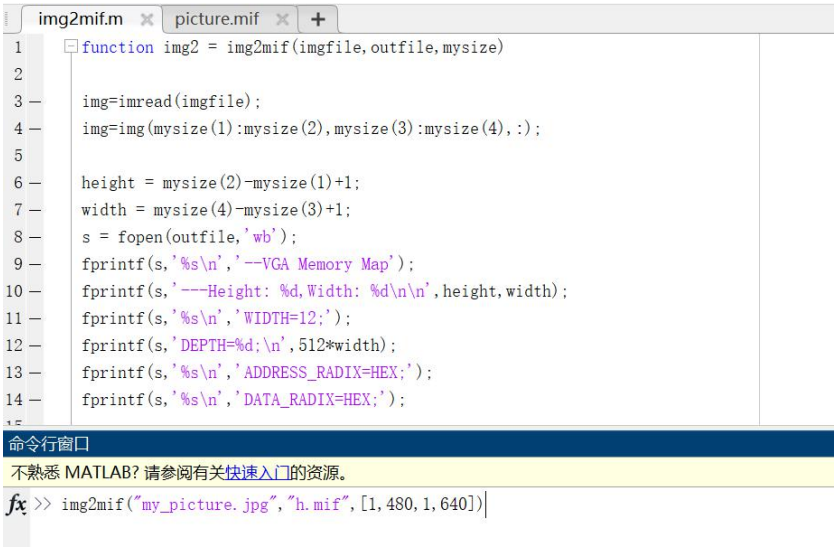
5.实验步骤

一．利用 systembuilder 建立工程

选择 CLOCK、SWITCH、LEDR、VGA

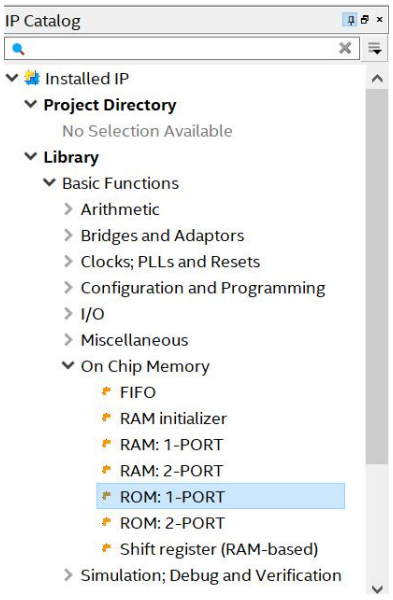


二．用 matlab 工程生成 mif 文件，键入相应命令即可

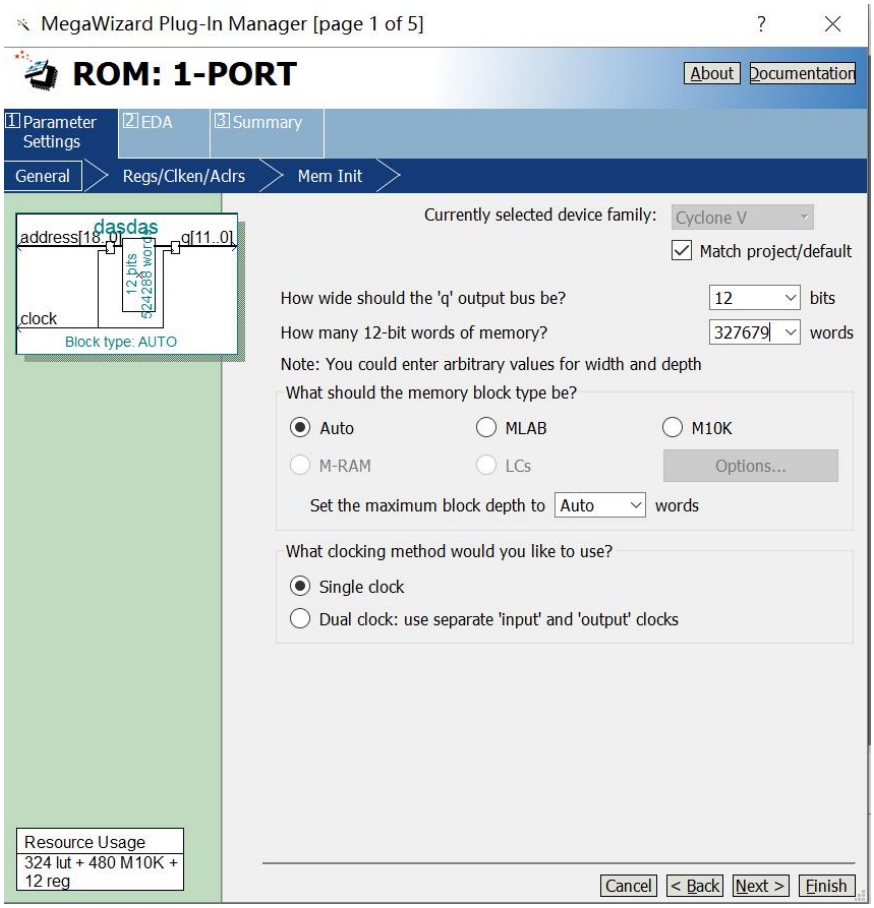


三．用 rom 存储 mif 文件，生成相应的.v 文件

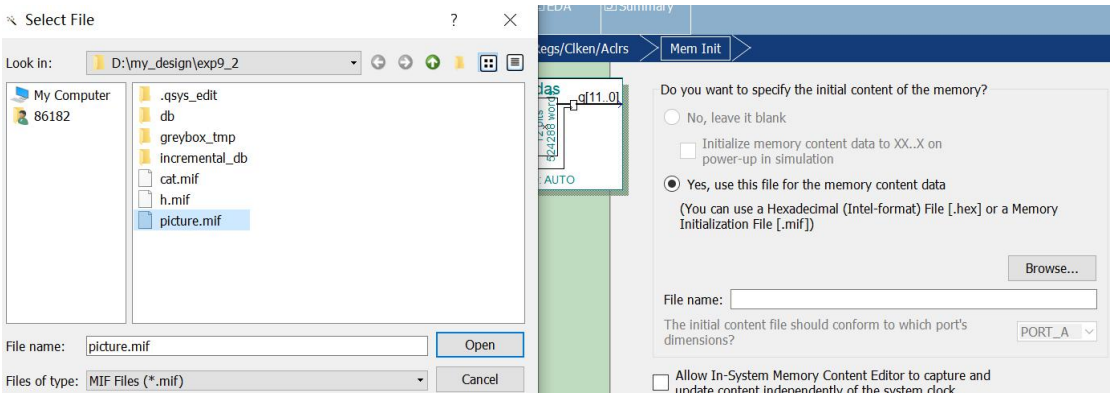
在右侧 IP catalog 下选择 ROM-1-PORT



每位选择 12bit，共有 327679 位



选择 mif 文件生成



四．编写 Verilog 代码

写一下相应模块的代码，在顶层文件中做相应的调用，进行编译调试，直至成功。

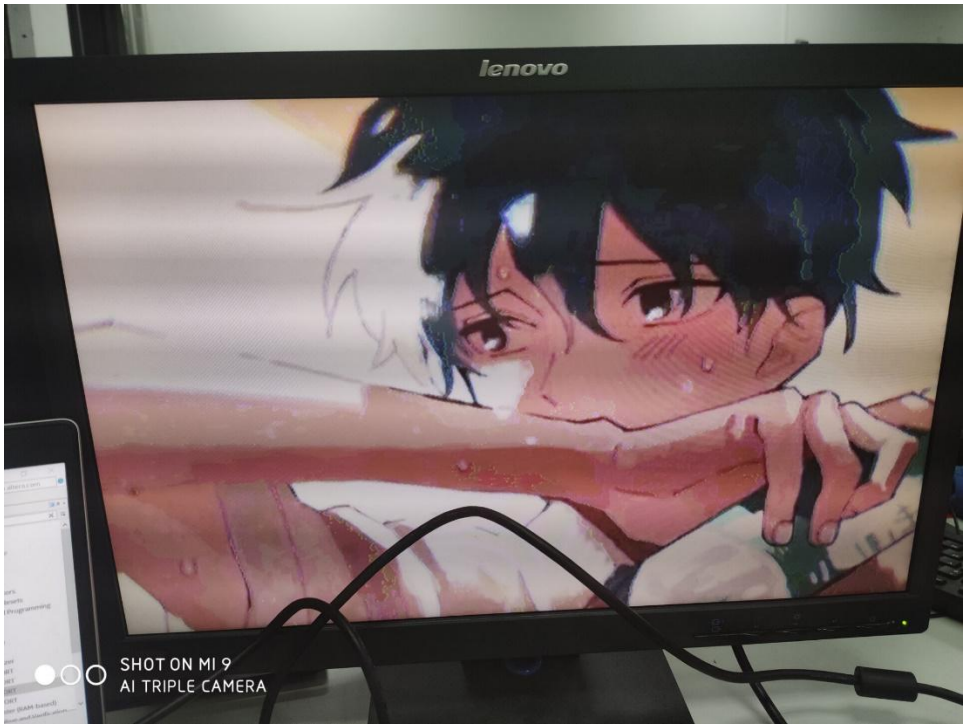
五．在板子和键盘上进行测试

6.测试方法

观察显示器的输出即可。

7.实验结果

图片：



8.实验中遇到的问题

1.matlab 指令不是很熟练，上网进行查询。一开始也没看明白 matlab 里那个函数要干嘛，后来经过学习一点 matlab 语法也明白了
2.不知道如何将给的 mif 文件弄到本地，后来直接复制过来再改个后缀就可以了。
3.一开始没有用 rom 存储，想直接用矩阵来存储，发现内存不够。

9.实验得到的启示

1．一定要学会用分模块编程，会让结构更加清晰，debug 更加容易
2．有问题可以向老师助教请教、和同学讨论、上网搜，很多时候是思路或结构出现了问题。
3. 有时可能真的是设备的问题。

10.意见和建议

1.这种实验测试文件不好测试，希望老师可以直接说清楚不用测试文件，只需要在板子上测试。
2.希望给一些实验原理的指导，不然自己搞有点懵。
3.感觉这样每次的实验都是接触的全新的东西，又无法深究，这样效率有点低。还是希望深究一点东西。